Kapitola: Číslovacie funkcie a veta o p. r. časovej zložitosti

Čo už vieme: Vo formalizme primitívnej rekurzie vieme definovať práve tie programy, ktoré vieme počítať programami s for-cyklami (bez while-cyklov, rekurzie a podobných zveriniek).

Otvorené otázky: Existujú nejaké ďalšie veci ekvivalentné primitívnej rekurzii? Iné pohľady, cez ktoré vieme povedať čo je a čo nie je primitívne rekurzívne? Sú vlastne na niečo dobré while cykly a všeobecná rekurzia? Sure, umožnia nám napísať program, ktorý pre niektoré vstupy neskončí, ale je to všetko? Aký je súvis medzi primitívnou rekurziou a napr. tým, čo poznáme z Foje ako rekurzívne jazyky? Ak nám niekto zaručí, že program vždy zastaví, musí nutne počítať primitívne rekurzívnu funkciu?

Cieľ: pomocou primitívne rekurzívnych funkcií vedieť zakódovať dvojicu prirodzených čísel do jedného. Chceme teda také primitívne rekurzívne funkcie $c(x, y)$, $l(x)$ a $r(x)$, aby platilo:

- c je prostá
- pre každé x, y je $l(c(x, y)) = x$
- pre každé x, y je $r(c(x, y)) = y$

Ako na to? Zoraďme si všetky dvojice prirodzených čísel do postupnosti. Systematicky to môžeme spraviť napr. tak, že ich zoradíme primárne podľa súčtu a sekundárne podľa prvého z nich:

$$(0, 0),\ (0, 1),\ (1, 0),\ (0, 2),\ (1, 1),\ (2, 0),\ (0, 3),\ (1, 2),\ \dots$$

Teraz $c(x, y)$ definujeme ako zero-based index dvojice (x, y) v tomto poradí.

Zjavne $c(x, y) = (1 + \cdots + (x + y)) + x = \frac{(x+y)(x+y+1)}{2} + x$, preto c je primitívne rekurzívna.

Použijeme veľký arzenál na dôkaz, že aj l je primitívne rekurzívna: Určite $\forall x : l(x) \leq x$ a tiež $\forall x : r(x) \leq x$. Preto stačí vyskúšať všetky (a, b) z rozsahu od $(0, 0)$ po (x, x) a nájsť tú dvojicu, ktorej kód je x.

Python:

```
def l(x):
  for a in range(0,x+1):
    for b in range(0,x+1):
      if c(a,b)==x:
        return a
```

Formálne to isté ide napr. cez dve vnorené primitívne rekurzie (koniec koncov, primitívna rekurzia je vlastne jednoduchý for-cyklus).

Príklad: Fibonacciho postupnosť je primitívne rekurzívna. Jeden možný dôkaz: spravíme si pomocnú funkciu $helper(n)$, ktorá bude vracať kód dvojice (F_n, F_{n+1}).

Python:

```
def helper(n):
  if n==0:
    return (0,1)
  else:
    x,y = helper(n-1)
    return (y,x+y)
```

Intuitívna rekurzívna definícia:

$$helper(0) = c(0, 1)$$
$$helper(n + 1) = c(\ r(helper(n)),\ l(helper(n)) + r(helper(n))\ )$$

$$fib(x) = l(helper(x))$$

A ešte pre názornosť úplne formálne (až na vynechanie definícií c, l a r):

$$\begin{aligned}
add &\equiv PR[P_1^1, Comp[s, P_1^3]] \\
j &\equiv Comp[s, z] \\
addpair &\equiv Comp[add, l, r] \\
almost &\equiv Comp[c, r, addpair] \\
g &\equiv Comp[almost, P_1^2] \\
helper &\equiv PR[j, g] \\
fib &\equiv Comp[l, helper]
\end{aligned}$$

Takto vieme kódovať do celého čísla ľubovoľnú konštantne veľkú n-ticu. Dá sa však ísť ešte ďalej. Poučení Leibnizom môžeme ľubovoľnú konečnú postupnosť ľubovoľnej dĺžky zakódovať do mocnín prvočísel. Napr. takto: Kódom $(a_1, \ldots, a_k)$ bude

$$p_1^{a_1} \cdots p_k^{a_k} p_{k+1}$$

kde p_i je i-te prvočíslo.

Funkciu $div(x, y)$ vieme definovať pre $y > 0$ nasledovne:

$$div(x, y) = \sum_{1 \le i \le x} \overline{sgn}(minus(iy, x))$$

Z nej ľahko definujeme mod, počet deliteľov, test prvočíselnosti, funkciu vracajúcu n-té prvočíslo a pomocou nich tiež funkcie na prácu s takto kódovanými postupnosťami.

Vieme reprezentovať pásku DTS ako kód konečnej postupnosti znakov, konfiguráciu DTS ako usporiadanú trojicu čísel (číslo stavu, pozícia hlavy, číslo kódujúce obsah pásky). Pre konkrétnu δ-funkciu DTS vieme definovať funkciu ktorá pre vstup=konfiguráciu vráti výstup=nasledujúcu konfiguráciu.

Veta o primitívne rekurzívnej časovej zložitosti: Nech existuje DTS A počítajúci funkciu $f(x)$, pričom existuje primitívne rekurzívna funkcia t taká, že $\forall n : A$ na vstupe n spraví nanajvýš $t(n)$ krokov. Potom f je primitívne rekurzívna.

Dôsledok: Ak vôbec existujú totálne Turingovsky vypočítateľné funkcie, musí ísť o funkcie, ktoré sa počítajú neuveriteľne ťažko.